ASSIGNMENT 3

Part 1: Controller Hub and Learning Switch

```
mininet@mininet-vm:~/A3$ ryu-manager controller_hub.py
loading app controller_hub.py
loading app ryu.controller.ofp_handler
```

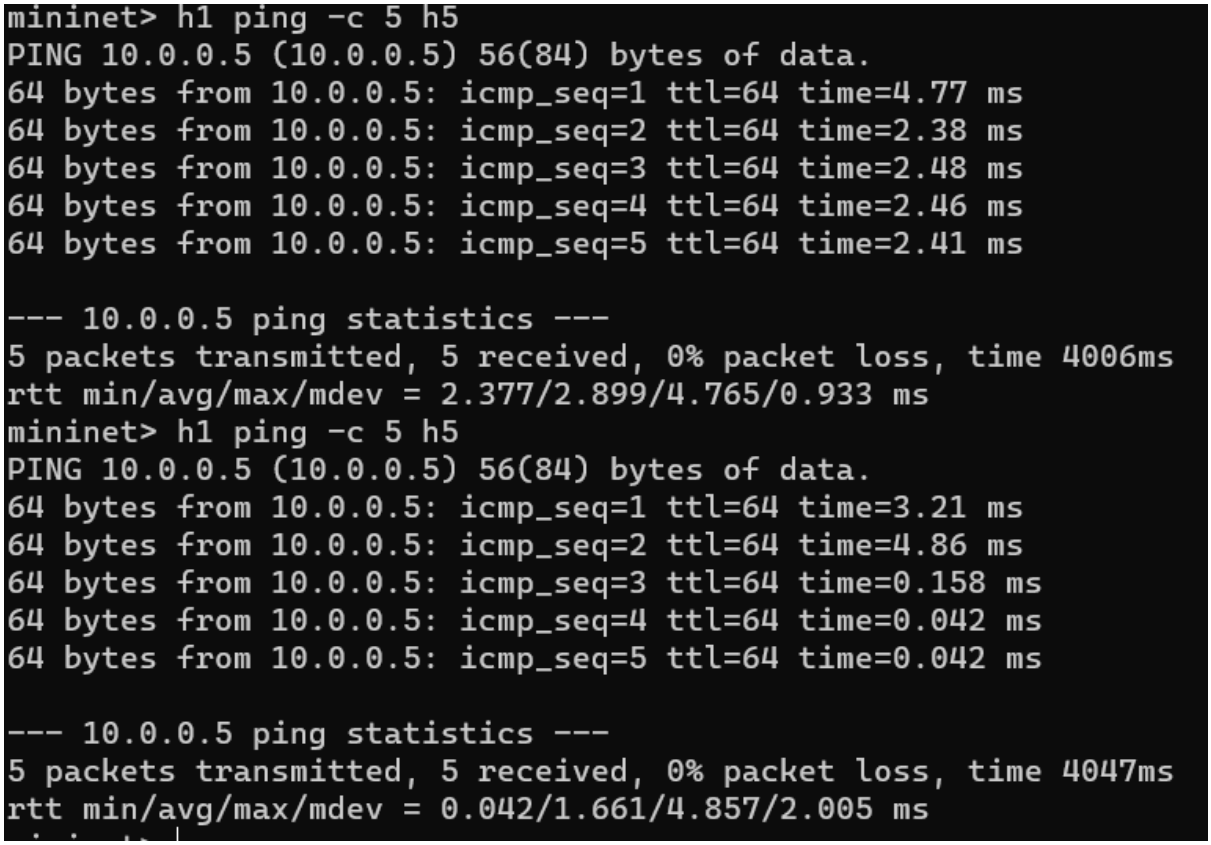
```
mininet> h2 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=2.48 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=2.31 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=2.47 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 2.309/2.417/2.476/0.076 ms
```

```
mininet@mininet-vm:~/A3$ ryu-manager learning_switch.py
loading app learning_switch.py
loading app ryu.controller.ofp_handler
```

```
mininet> h2 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=5.75 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=2.22 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.200 ms
```

In this case, the latencies of the three ping packets with the Hub controller are nearly identical, but the learning switch minimises delay by learning from prior packets.

The observed latency differences between a Hub Controller and a Learning Switch primarily arise from their distinct roles and functions: the Hub Controller introduces additional processing and communication overhead due to its centralized control and policy management in Software-Defined Networking (SDN), while the Learning Switch minimizes latency by efficiently forwarding traffic based on its dynamically updated MAC address table in traditional Ethernet networks.



Throughput

the Learning Switch, often used in SDN, maintains a MAC address table to intelligently forward packets based on destination addresses, which reduces unnecessary broadcast traffic. This approach is more efficient in terms of throughput as it only sends packets to the intended destination, making it suitable for larger, less congested networks. However, it requires more processing overhead to maintain the MAC address table and may still have limitations in handling very high traffic loads. The choice between these two methods depends on network size, traffic patterns, and performance requirements.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet>
```

Part 2: Firewall and Monitor

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X h5
h2 -> h1 h3 h4 X
h3 -> h1 h2 h4 X
h4 -> X h2 h3 h5
h5 -> h1 X X h4
*** Results: 30% dropped (14/20 received)
```

Here as operator wants we prevent communication between H2 and H3 with H5, and H1 with H4. Installed rule is to drops any packets with a source IP address or MAC address matching one host and a destination IP address or MAC address matching the other host, this will effectively blocks their communication.

To ensure that pre-existing rules do not interfere with the real-time firewall policy, the network operator should assign higher priority levels to firewall rules, follow a well-defined rule evaluation order, and implement conflict detection and resolution mechanisms, while continuously monitoring and auditing rule changes to maintain consistency.

Part 3: Load Balancer

```
mininet@mininet-vm:~/A3$ ryu-manager load_balancer.py
loading app load_balancer.py
loading app ryu.controller.ofp_handler
instantiating app load_balancer.py of SimpleSwitch13
Done with initial setup related to server list creation.
instantiating app ryu.controller.ofp_handler of OFPHandler
The selected server is ===>  10.0.0.4
<========Packet from client: 10.0.0.1. Sent to server: 10.0.0.4, MAC: 00:00:00:00:00:04 and on switch port: 1========>
<++++++++Reply sent from server: 10.0.0.4, MAC: 00:00:00:00:00:04. Via load balancer: 10.0.0.42. To client: 10.0.0.1++++++++>
(((Entered the ARP Reply function to build a packet and reply back appropriately)))
{{{Exiting the ARP Reply Function as done with processing for ARP reply packet}}}
::::Sent the packet_out::::
The selected server is ===>  10.0.0.5
<========Packet from client: 10.0.0.1. Sent to server: 10.0.0.5, MAC: 00:00:00:00:00:05 and on switch port: 2========>
<++++++++Reply sent from server: 10.0.0.5, MAC: 00:00:00:00:00:05. Via load balancer: 10.0.0.42. To client: 10.0.0.1++++++++>
The selected server is ===>  10.0.0.4
<========Packet from client: 10.0.0.1. Sent to server: 10.0.0.4, MAC: 00:00:00:00:00:04 and on switch port: 1========>
<++++++++Reply sent from server: 10.0.0.4, MAC: 00:00:00:00:00:04. Via load balancer: 10.0.0.42. To client: 10.0.0.1++++++++>
(((Entered the ARP Reply function to build a packet and reply back appropriately)))
{{{Exiting the ARP Reply Function as done with processing for ARP reply packet}}}
::::Sent the packet_out::::
The selected server is ===>  10.0.0.5
<========Packet from client: 10.0.0.2. Sent to server: 10.0.0.5, MAC: 00:00:00:00:00:05 and on switch port: 2========>
<++++++++Reply sent from server: 10.0.0.5, MAC: 00:00:00:00:00:05. Via load balancer: 10.0.0.42. To client: 10.0.0.2++++++++>
The selected server is ===>  10.0.0.4
<========Packet from client: 10.0.0.2. Sent to server: 10.0.0.4, MAC: 00:00:00:00:00:04 and on switch port: 1========>
<++++++++Reply sent from server: 10.0.0.4, MAC: 00:00:00:00:00:04. Via load balancer: 10.0.0.42. To client: 10.0.0.2++++++++>
(((Entered the ARP Reply function to build a packet and reply back appropriately)))
{{{Exiting the ARP Reply Function as done with processing for ARP reply packet}}}
::::Sent the packet_out::::
```

Every time we ping to 10.0.0.42 the requests are evenly distributed between sever H4 and H5 in round robin. To implement a load balancing policy that considers the load on servers, I would add periodic monitoring of server performance and dynamically distribute incoming traffic to servers with lower loads, ensuring efficient resource utilization and improved user experience.